PROPUESTA CURRICULAR PARA LA IMPLEMENTACION DE UN LABORATORIO DE SISTEMAS OPERATIVOS MEDIANTE EL USO DE LAS HERRAMIENTAS LIBRES MINIX 3, NACHOS Y LA PROGRAMACION POR HILOS

DIEGO RAMIREZ GOMEZ

CESAR AUGUSTO MORALES GOMEZ

INGENIERO

JUAN DE JESUS VELOZA MORA

UNIVERSIDAD TECNOLOGICA DE PEREIRA
FACULTAD DE INGENIERIA
INGENIERIA DE SISTEMAS Y COMPUTACION
PEREIRA
2018

PROPUESTA CURRICULAR PARA LA IMPLEMENTACION DE UN LABORATORIO DE SISTEMAS OPERATIVOS MEDIANTE EL USO DE LAS HERRAMIENTAS LIBRES MINIX 3, NACHOS Y LA PROGRAMACION POR HILOS

Proyecto presentado como requisito
Para optar el título de Ingeniera Sistemas y Computación

DIRECTOR
INGENIERO
JUAN DE JESUS VELOZA MORA

UNIVERSIDAD TECNOLOGICA DE PEREIRA
FACULTAD DE INGENIERIA
INGENIERIA DE SISTEMAS Y COMPUTACION
PEREIRA
2018

“La gente con conocimientos técnicos está dispuesta a perdonar a un ordenador que se cuelga un par de veces al año, pero los usuarios normales no”

Andrew S. Tanenbaum

CONTENIDO

LISTA DE FIGURAS

1. INTRODUCCION

1.1. RESUMEN

Este documento está realizado con el fin de crear una propuesta curricular de un área de aprendizaje donde se puedan fortalecer e implementar todos los conceptos que son adquiridos en la materia Sistemas Operativos I, del pensum actual de Ingeniería de Sistemas y Computación de la Universidad Tecnológica de Pereira, apoyados en herramientas libres como Minix 3 y NachOS, además de la programación por hilos de ejecución, un área de conocimiento bastante amplia y necesaria en nuestra formación como ingenieros.

1.2. ABSRACT

This document is made with the purpose of creating a curricular proposal of a learning area, where all the concepts that are acquired in the subject Operating Systems I can be strengthened and implemented, from the current curriculum of Systems Engineering and Computer Science of the Technological University of Pereira, supported by free tools such as Minix 3 and NachOS, in addition to the programming by threads of execution, an area of knowledge quite wide and necessary in our training as engineers.

1.3. INTRODUCCION

Los sistemas operativos nacieron de la necesidad de interacción entre el usuario y la máquina, es el encargado de administrar de manera óptima los recursos del hardware. Para nosotros como Ingenieros de Sistemas y Computación, es de vital importancia conocer y aplicar conocimientos básicos de Sistemas Operativos. El propósito de este proyecto es crear una propuesta de fortalecimiento de los conocimientos adquiridos en la materia sistemas operativos I, mediante el uso de prácticas en sala, preparando al estudiante con anterioridad para la enseñanza de los temas de la materia sistemas distribuidos. Es importante que los estudiantes fortalezcan y apliquen el manejo de procesos en sistemas operativos.

2. GENERALIDADES

2.1. PLANTEAMIENTO DEL PROBLEMA

En el pensum actual de la facultad de ingeniería de sistemas y computación de la Universidad Tecnológica de Pereira no existe un laboratorio, donde se permitan fortalecer los conocimientos adquiridos en la materia sistemas operativos I y que sirva como preparación para la materia sistemas distribuidos.

2.2. JUSTIFICACION

Los sistemas operativos, aunque son usados a diario por la mayoría de nosotros, en ocasiones son bastante subestimados, y nos enfocamos más en las tecnologías que usamos sobre ellos, mas no en el sistema operativo como tal, que es el que nos permite la interacción directa entre usuario y máquina, para nosotros como ingenieros de sistemas y computación, los sistemas operativos deben ser una de las áreas fundamentales del conocimiento, conocer sus ventajas, desventajas, el cómo ellos nos pueden ayudar brindándonos un amplio catálogo de herramientas para crear soluciones tanto en la vida personal, como en la laboral, por eso traemos este documento como propuesta para fortalecer el área de conocimiento en sistemas operativos modernos, que ya encontramos una asignatura en el pensum de ingeniería de sistemas y computación de nuestra universidad, que nos da los conceptos, una asignatura más enfocada a lo teórico, que a la practico. El que un estudiante obtenga un conocimiento práctico en los fundamentos que se le brindan en la materia teórica, le darán un plus sobre la competencia en el mundo laboral, le brindará herramientas, y una mayor abstracción de los conceptos para su vida laboral, realzará el programa, como uno de los que están comprometidos con que la universidad resalte como una institución comprometida con el conocimiento llevado también a lo práctico.

2.3. OBJETIVOS

2.3.1. OBJETIVO GENERAL

Elaborar una propuesta curricular para la implementación de un laboratorio de la materia sistemas operativos I, mediante el uso de las herramientas libres minix3, NachOS, y la programación con hilos, donde los estudiantes puedan reforzar y llevar a la práctica los conocimientos teóricos adquiridos en esta materia.

2.3.2. OBJETIVOS ESPECIFICOS

- Analizar las herramientas propuestas para la implementación del laboratorio.
- Elaborar el diseño de los laboratorios a desarrollar.
- Implementar una guía general para la implementación de los laboratorios.

3. MARCO TEORICO

3.1. HERRAMIENTAS PROPUESTAS

3.1.1. MINIX 3

3.1.1.1. INTRODUCCION

Es un Sistema operativo pequeño, monolítico, basado en Unix, desarrollado por Andrew Stuart Tanenbaum, el autor del libro *"Sistemas Operativos Análisis y Diseño"*, propuesto como referencia bibliográfica para la materia Sistemas Operativos I. Es un sistema operativo orientado a ser de alta disponibilidad, con baja intervención del usuario en la resolución de fallas. Es un sistema operativo que podemos considerar liviano, ya que maneja una cantidad mínima de código de ejecución en el kernel, además de manejar como procesos independientes el servidor de archivos, procesos y los controladores de hardware.

Los errores en este sistema operativo pueden ser más fáciles de encontrar comparado con otros sistemas con cantidades de código exuberantes en el kernel, incluso a veces innecesarios.

El hecho de que maneje los procesos de controladores de hardware como procesos independientes en el kernel, garantiza un plus de seguridad en la continuidad de ejecución, ya que los códigos externos necesarios para que el hardware funcione son desconocidos,

Es un software de código abierto, esto nos llevó a proponer este sistema operativo como herramienta para la creación de esta propuesta curricular.

3.1.1.2. POLÍTICAS DE CONFIABILIDAD

Minix 3 busca garantizar la fiabilidad al usuario, buscando centrarse en varios principios, sus desarrolladores se basan en estos aspectos para publicitarlo.

3.1.1.3. REDUCCIÓN DE CÓDIGO EN EL KERNEL

Como podemos imaginar, la competencia actual de los desarrolladores de sistemas operativos por liderar el campo de ser los más eficientes, es una batalla continua, obligados constantemente a innovar y generar nuevas funcionalidades y/o mejoras que busquen resaltar su producto por encima de los demás, esto conlleva a grandes cantidades de código ejecutándose y operando como un gran reloj, donde por supuesto, la cantidad de código afecta directamente a la probabilidad de fallo, fallos que aprovechan los usuarios maliciosos para atacar el sistema.

Para nosotros como usuarios es algo maravilloso, ya que nos abre un catálogo cuantioso de sistemas orientados a la función de interacción entre usuario y máquina, cada uno con sus ventajas y desventajas. Pero en nuestro objetivo académico, debemos buscar la simplicidad y objetividad; Minix 3 maneja una cantidad aproximada de 4000 líneas de código ejecutable en su kernel, buscando como se mencionó anteriormente que la cantidad de código sea directamente proporcional a la probabilidad de fallo.

3.1.1.4. CONTROLADORES DE HARDWARE COMO PROCESOS INDEPENDIENTES

En la mayoría de los sistemas operativos monolíticos, los controladores de dispositivos son ejecutados en su núcleo, es decir que todos los dispositivos hardware que son agregados, conllevan a integrar código de fuentes en ocasiones no confiables en el kernel, lo que conlleva por supuesto, a una reducción de seguridad considerable, Además del incremento en la probabilidad de fallo, con tan solo una línea de código que falle, derribara todo nuestro proceso en el kernel.

El sistema operativo que traemos como propuesta maneja cada controlador de dispositivo como procesos independientes, es decir que tienen una reducción considerable de permisos que tendría en cambio si fuera ejecutado en el kernel, no pueden ejecutar instrucciones privilegiadas, ni escribir en memoria absoluta.

3.1.1.5. LIMITES AL ACCESO A MEMORIA PARA LOS CONTROLADORES

En los sistemas operativos monolíticos, los controladores pueden escribir libremente en memoria, lo que significa una grave falla de seguridad, conlleva a tener un sistema susceptible a fallas, el sistema operativo que estamos analizando crea un descriptor encargado de aprobar o denegar accesos a estos controladores, además de dar dirección a estos accesos. Los controladores le piden al kernel poder escribir a través de este descriptor, lo que reduce considerablemente la probabilidad de fallas en este aspecto ya que no se les da permisos de escribir fuera del kernel

3.1.1.6. SOBREVIVIR A LOS MALOS PUNTEROS

Los errores en programación son fatales en muchos sistemas operativos, y dentro de estos posibles errores de programación, están los punteros erróneos, en Minix 3 un puntero erróneo bloqueara el proceso del controlador con el código erróneo, evitando así que tenga algún efecto sobre el sistema operativo como un todo, el reincarnation server, que mencionaremos más

adelante, reiniciará el controlador bloqueado, este procesos será en algunos controladores transparente para el usuario.

3.1.1.7. BUCLES INFINITOS

En la mayoría de sistemas operativos monolíticos, cuando un controlador entra en un bucle infinito, el sistema operativo queda bloqueado, Minix 3 cuando detecta que el código de un controlador entra en un bucle, reducirá gradualmente su prioridad, hasta llegar a ser un proceso inactivo, luego el rencarnation server verá que este proceso no responde a las solicitudes de estado, y por consiguiente bloqueara y reiniciará el controlador.

3.1.1.8. LIMITAR EL DAÑO DE LOS REBASAMIENOS DE BUFER

En Minix 3 se usan mensajes de longitud fija para la comunicación interna, reduciendo problemas en la administración de búfer; Además, muchos ataques funcionan superponiendo un búfer para engañar al programa para que regrese de una llamada de función utilizando una dirección de retorno apilada sobrescrita que apunta hacia el búfer de saturación. En este sistema operativo se mitiga este tipo de ataque ya que la instrucción y el espacio de datos están divididos y solo se puede ejecutar el código en el espacio de instrucción.

3.1.1.9. RESTRICCION AL ACCESO DE LAS FUNCIONES DEL KERNEL

Los controladores obtienen servicios del kernel como copiar datos en los espacios de direcciones de los usuarios haciendo llamadas al kernel, en núcleos monolíticos, cada controlador puede llamar a cada función del kernel libremente. En Minix 3, el kernel usa un mapa de bits que especifica que llamadas está autorizado a realizar cada controlador.

3.1.1.10. RESTRICCION A LOS PUERTOS DE ENTRADA Y SALIDA

El kernel de este sistema operativo también cuenta con una tabla que indica a cada controlador a que puertos de E/S tienen permiso de acceder, garantizando que solo puedan acceder a sus

propios puertos de E/S. En el caso de los sistemas operativos monolíticos, los controladores pueden acceder a puertos de E/S de otros dispositivos.

3.1.1.11. RESTRICCION A LA COMUNICACIÓN ENTRE COMPONENTES

Minix 3 cuenta también con un mapa de bits determinado a procesos, determinando los destinos a los que puede enviar cada proceso, mitigando así la mala administración de la comunicación entre componentes, ya que no todos se pueden comunicar entre ellos.

3.1.1.12. REINCARNATION SERVER

Esta es una de las características de este sistema operativo que lo llevan a resaltar entre sus competidores, el rencarantion server, es un proceso que se encarga de enviar pings constantemente a los controladores de dispositivos, y aquel que no responda a estos llamados, el sistema lo toma como una señal de que este controlador está fallando, así que reinicia este proceso. Toda esta operación es invisible para el usuario, es decir que mientras usamos el sistema operativo, este mismo esta auto examinándose constantemente y reiniciando procesos caídos.

3.1.1.13. INTEGRACION DE INTERRUPCIONES Y MENSAJES

Cuando se produce una interrupción, se envía una notificación al controlador apropiado. Si el conductor está esperando un mensaje, obtiene la interrupción inmediatamente; de lo contrario recibe la notificación la próxima vez que hace una solicitud para obtener un mensaje. Este esquema elimina las interrupciones anidadas y facilita la programación del controlador.

3.1.1.14. INSTALACION

La siguiente guía de instalación fue obtenida de la página oficial de Minix 3[1].
La imagen del sistema operativo se puede obtener desde el siguiente enlace http://wiki.minix3.org/doku.php?id=www:download:start
Los requisitos mínimos de hardware se pueden ver en el siguiente enlace

http://wiki.minix3.org/doku.php?id=usersguide:hardwarerequirements

La siguiente guía de instalación está diseñada para procesadores Intel, para arquitecturas ARM se recomienda visitar la guía en la página oficial de Minix 3 para ARM [2].
La imagen de instalación se puede obtener directamente desde la página web de Minix 3 [3]
En el momento de que surge el inicio de sesión, se debe iniciar como root, el sistema solicitará una contraseña, la cual por defecto viene en blanco, así que solo se debe presionar la tecla "ENTER" para continuar. Para comenzar la instalación en el disco duro, se debe ejecutar el script "setup",

3.1.1.15. ARQUITECTURA

El sistema operativo Minix 3 cuenta con una arquitectura en capas como se muestra en la figura 1, esta arquitectura divide el sistema en una serie de niveles dedicados a funciones específicas, aunque algunas funciones, dependen de las funciones ejecutadas en otras capas.

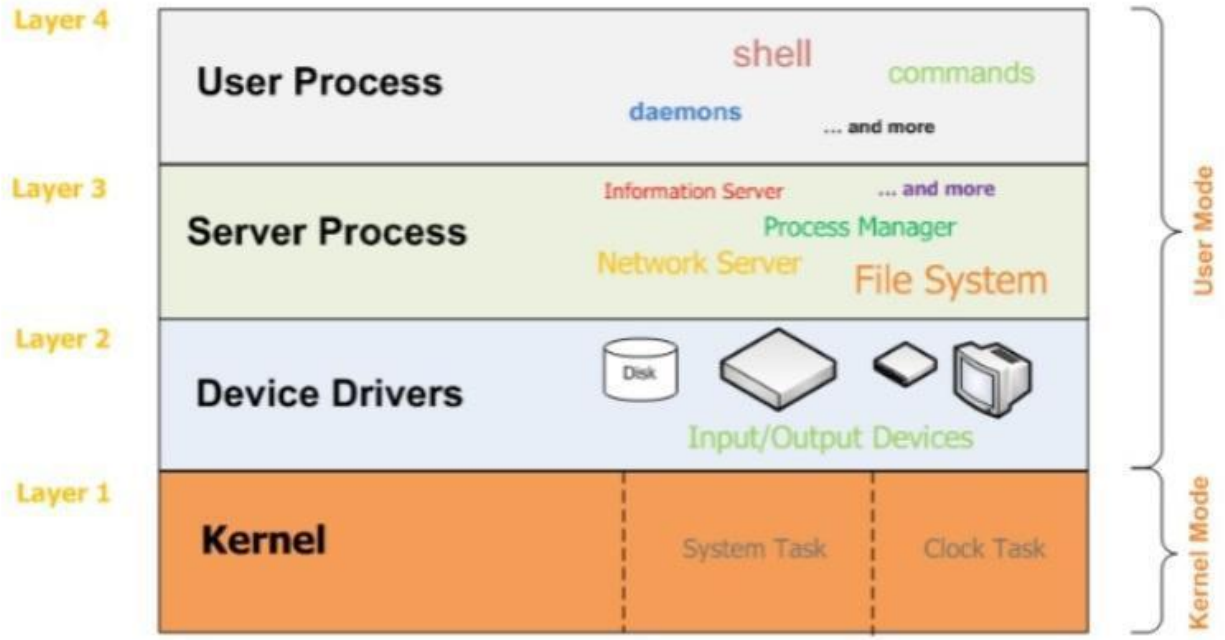

Figura 1: Arquitectura de Minix 3 [4]

En este sistema operativo la mayoría de las funcionalidades clave de él se implementan y ejecutan separadamente del kernel. Este diseño hace que el sistema operativo sea modular y extensible, puesto que es posible desarrollar nuevos servicios con relativamente pocos cambios en el kernel. Los servicios fundamentales que provee un microkernel son la gestión del espacio de direcciones, gestión de hilos, comunicación entre procesos y gestión de los temporizadores. [4]
El objetivo de la arquitectura es lograr una alta fiabilidad ejecutando casi todo como procesos en modo usuario.

3.1.1.16. CAPA DEL KERNEL

Esta capa provee las funcionalidades de más bajo nivel necesarias para la ejecución del sistema, entre los que podemos encontrar gestión de interrupciones, guardar y restaurar registros, planificar procesos, ofrecer servicios a capas superiores, funciones de comunicación y mensajes. Las funcionalidades de esta capa que están ligadas directamente con el hardware, están escritas en lenguaje ensamblador, el resto de funciones están escritas en lenguaje C.

3.1.1.17. CONTROLADORES DE DISPOSITIVOS

En esta capa, se encuentra todo el código escrito con el fin de dar manejo a las tareas de E/S de los distintos controladores, incluyendo periféricos del equipo como teclado, parlantes, mouse, etc., Además también se encarga de la tarea de dar soporte a ciertas tareas que no pueden ser realizadas a nivel de usuario

3.1.1.18. PROCESOS DE SERVIDOR EN MODO USUARIO

En la parte superior del micro-kernel se están ejecutando funciones de servidor que se encargan de ofrecer servicios al resto del sistema. Estos procesos carecen de la autoridad para acceder directamente al hardware o software que no les pertenece, debido a que se ejecutan en modo de usuario, utilizan llamados al sistema para interactuar con otros procesos y dispositivos de hardware a través del micro-kernel

Un dato importante que cabe resaltar, es que en Minix 3 todos los procesos son ejecutados como procesos de usuario, excepto el micro-kernel.

3.1.1.19. PROCESOS DE USUARIO

En esta etapa se ejecutan todos aquellos procesos de usuario, en esta etapa podemos encontrar terminales, controladores, y cualquier otro proceso que el usuario desee ejecutar, esta capa, para cumplir con esta tarea, se basa en todas las capas que se encuentra por debajo de ella, como se ilustra en la Figura 1, es decir, cuando el usuario realiza una petición al sistema, esta capa busca apoyo en sus sucesoras para realizar esta tarea.

3.1.1.20. MANEJO DE PROCESOS

Un sistema operativo multi-programado gestiona las peticiones de E/S de los procesos mediante interrupciones, bloqueando los procesos en el momento que realizan peticiones de E/S, dando así espacio para que otros procesos sean ejecutados. Cuando esta petición se ha resuelto, el proceso en ejecución es interrumpido por cualquier pieza de hardware, y deja de estar activo mientras el dispositivo atiende su petición. Es tarea de las capas más bajas ocultar esas interrupciones transformándolas en mensajes. En el momento que el proceso de E/S termina envía un mensaje que activa a algún proceso poniéndolo listo para ejecución. El reloj también puede generar interrupciones, garantizando que un proceso que no ha realizado entrada/salida libere la CPU en algún momento y permita la ejecución de otros procesos. [4]

Minix 3 utiliza una estructura de varios niveles de colas, cada una con distinta prioridad. Se definen dieciséis colas por defecto, aunque puede recompilarse para definir más o menos colas según se desee. La cola de menor prioridad es utilizada únicamente por el proceso IDLE, que se ejecuta cuando no hay otros en cola. Los procesos de usuario comienzan por defecto en una cola de varios niveles de prioridad, distinta a la más baja.

Existen dos estructuras básicas que usa el Gestor de Procesos: La tabla de procesos (mantenida en la variable “mproc”) y la tabla de huecos. Algunos de los campos de la tabla de procesos los necesita el “kernel”, otros el Gestor de Procesos y otros el Gestor de Ficheros. En Minix 3, cada una de estas tres partes del S.O. tiene su propia tabla de procesos, en la que únicamente se tienen los campos que necesitan. Salvo algunas excepciones, las entradas en la tabla se corresponden exactamente. Así la posición ‘k’ de la tabla del G.P. se refiere al mismo proceso que la posición ‘k’ de la tabla del G.F.. Cuando se crea o destruye un proceso, las tres partes actualizan sus tablas para reflejar la nueva situación consistentemente. Las excepciones son procesos que no son conocidos fuera del “kernel”, como las Tareas de Reloj y de Sistema, o bien los “falsos” procesos IDLE y KERNEL (ocupan entrada pero no tienen código). En la tabla del “kernel”, estas excepciones tienen asignados números negativos de entrada. Estas entradas no existen ni en el G.P. ni en el G.F.. Así, lo dicho anteriormente acerca de la posición ‘k’, es estrictamente cierto para valores mayores o iguales a cero. Otros procesos de Minix como el Gestor de Procesos y el Gestor de Ficheros tienen asignadas las entradas 0 y 1 respectivamente en todas las tablas

3.1.1.21 MANEJO DE HILOS

Los hilos son similares a los procesos ya que ambos representan una secuencia simple de instrucciones ejecutada en paralelo con otras secuencias. Los hilos son una forma de dividir un programa en dos o más tareas que corren simultáneamente, compitiendo, en algunos casos, por la CPU. La diferencia más significativa entre los procesos y los hilos, es que los primeros son típicamente independientes, llevan bastante información de estados, e interactúan sólo a través de mecanismos de comunicación dados por el sistema. Por otra parte, los hilos generalmente comparten la memoria, es decir, acceden a las mismas variables globales o dinámicas, por lo que no necesitan costosos mecanismos de comunicación para sincronizarse. Por ejemplo un hilo podría encargarse de la interfaz gráfica (iconos, botones, ventanas), mientras que otro hace una larga operación internamente. De esta manera el programa responde más ágilmente a la interacción con el usuario. En sistemas operativos que proveen facilidades para los hilos, es más rápido cambiar de un hilo a otro dentro del mismo proceso, que cambiar de un proceso a otro. Es posible que los hilos requieran de operaciones atómicas para impedir que los datos comunes sean

cambiados o leídos mientras estén siendo modificados. El descuido de esto puede generar estancamiento.

Ejemplo:

```
#include <pthread.h>
int pthread_create(pthread_t *tid,
                   const pthread_attr_t *attr,
                   void *(*star_routine)(void *),
                   void *arg)

void pthread_exit(void *value_ptr);

int pthread_join(pthread_t thread, void **value_ptr);
pthread_t pthread_self(void);
```

Los hilos se crean dinámicamente (es decir se pueden crear en cualquier instante durante la ejecución de un proceso) con la función **pthread_create**, la cual crea un hilo y lo coloca en una cola de hilos preparados.

pthread_t *tid Esto indica que como argumento se debe colocar la dirección de una variable de tipo pthread_t, y si la llamada tuvo éxito en ella se almacenará el identificador del hilo. En caso que se desee crear varios hilos, se puede emplear un arreglo para almacenar los identificadores.

const pthread_attr_t *attr Los atributos del hilo se encapsulan en el objeto atributo al que apunta attr. Si attr es NULL, el nuevo hilo tendrá los atributos por omisión.

void *(*star_routine)(void *) El tercer argumento, start_routine es el nombre de una función a la que el hilo invoca cuando inicia su ejecución. La función debe devolver un puntero sin tipo (void *) y solo puede tener un argumento que debe ser un puntero sin tipo. Esto no es una limitación, sino una ventaja, ya que se puede emplear como argumento cualquier tipo y después la función recuperar el tipo haciendo un *cast*.

void *arg El último argumento es la dirección de alguna variable que se desea pasar como parámetro de la función.

La función **pthread_exit** termina el hilo que la invoca. El valor del argumento value_ptr queda disponible para pthread_join si ésta tuvo éxito. Sin embargo, el value_ptr en pthread_exit debe apuntar a datos que existan después que el hilo ha terminado, así que no puede asignarse como datos locales automáticos para el hilo que está terminando. La función pthread_exit invoca controladores de terminación de hilos, cosa que return no hace.

La función **pthread_join** suspende la ejecución del hilo invocador hasta que el hilo identificado con thread termine, ya sea porque llamó a pthread_exit o por que fue cancelado. Esta llamada es similar a waitpid en el nivel de procesos. Si value_ptr no es NULL, entonces el valor retornado por thread es almacenado en la ubicación apuntada por value_ptr.

La función **pthread_self** devuelve el identificador del hilo que está invocando la función.

3.1.1.22 MANEJO DE MEMORIA

La gestión de memoria en Minix 3 es simple. En primer lugar, no se utiliza paginación en absoluto y en segundo lugar, aunque el código fuente completo incluye la posibilidad del uso de intercambio de memoria ("swapping"), por simplicidad, no se estudiará esta opción, ya que en la actualidad, la cantidad de memoria disponible en los sistemas, hace que en la práctica, ésta sea raramente necesaria. El Gestor de Procesos ("Process Manager") es el proceso de Minix que se encarga de gestionar la memoria así como de manejar las llamadas al sistema relacionadas con la gestión de procesos. Algunas como "fork, exec y brk" muy relacionadas con la gestión de memoria, otras relacionadas con señales, y otras con características de los procesos, como el usuario y grupo propietario, tiempos de uso, etc. En cuanto a las estructuras de datos gestionadas, destacan la Tabla de Procesos, con sus campos específicos y la Lista de huecos, la cual mantiene los huecos de memoria libres, ordenados ascendentemente por dirección de memoria. Sin el uso de Intercambio, la posición en memoria de un proceso no cambia durante toda su ejecución y tampoco aumenta o disminuye su asignación de espacio.
En las operaciones normales de MINIX 3 se asigna memoria a un proceso sólo en dos ocasiones: Con la llamada "fork", en la que asigna la cantidad de memoria que precisa el hijo y con la llamada "execve", en la que se devuelve la memoria usada por la vieja imagen a la lista de huecos

libres y se asigna memoria para la nueva imagen. La liberación de memoria ocurre en dos ocasiones: a) Cuando un proceso muere, bien sea por la llamada “exit” o la recepción de una señal y b) Durante la llamada “execve” si ésta tiene éxito. En la llamada “fork”, dependiendo de si los espacios I+D son juntos o separados, se asigna memoria para el proceso hijo de una forma u otra. Con espacios juntos, la memoria nueva asignada constituirá un solo bloque (segmento de Intel), del mismo tamaño que el del padre, el cual se corresponde con la memoria total utilizada por el padre, es decir, la suma de los tamaños del código, datos, gap y pila. Con espacios separados, se asigna un nuevo bloque para el hijo de igual tamaño que el de padre, pero este bloque (segmento) contiene únicamente las áreas de datos, gap y pila. El bloque de código (segmento) en cambio, no se asigna, ya que puede ser y es, compartido con el padre. En la llamada “execve” se asigna al proceso la cantidad de memoria indicada en la cabecera del fichero ejecutable, teniendo en cuenta si los espacios son juntos o separados, tal y como ya se ha contado. Hay que tener en cuenta en esta llamada, que si los espacios son separados se complica un poco la gestión, ya que al liberar la vieja imagen, tendrá que tenerse en cuenta si el bloque de código está siendo compartido por otro proceso, en cuyo caso no se liberaría éste último. Este problema no ocurre con espacios juntos, ya que al no haber bloques de código compartidos, siempre que se libera la imagen de memoria vieja, se libera de forma completa, ya que ésta constituye un único segmento (o bloque).

3.1.2. NachOS

3.1.2.1. INTRODUCCION

Es un sistema operativo bastante apropiado para nuestros objetivos, fue creado específicamente para uso educativo, por eso decidimos agregarlo como herramienta para nuestra propuesta curricular. Fue desarrollado por los señores Wayne A. Christopher, Steven J. Procter, y Thomas E. Anderson, en Berkley, en la Universidad de California, en 1991.

Este sistema operativo es ejecutado como un proceso de usuario en el sistema operativo anfitrión, es decir que no se puede decir completamente que sea un sistema operativo como tal, fue escrito

en C++. Un simulador de MIPS ejecuta el código para cualquier programa de usuario que se ejecute sobre el sistema operativo NachOS [5]

3.1.2.2. GESTION DE HILOS

En Nachos como en la mayoría de sistemas operativos, un proceso consiste en:

- Un espacio de direcciones, el espacio de direcciones incluye toda la memoria a la que se puede referenciar el proceso. En algunos sistemas, dos o más procesos pueden compartir parte de un espacio de direcciones, pero en los sistemas tradicionales el contenido de un espacio de direcciones es privado de ese proceso. El espacio de direcciones se divide a su vez en:
    - Código ejecutable, como las instrucciones del programa
    - Espacio de pila para variables locales
    - Espacio heap, para variables globales y memoria asignada dinámicamente (por ejemplo, como la obtenida por el malloc Unix o el nuevo operador C++). En Unix, el espacio heap se desglosa en BSS (contiene variables inicializadas a 0) y secciones DATA (variables inicializadas y otras constantes).[7]
- Un único hilo de control, por ejemplo, la CPU ejecuta las instrucciones secuencialmente dentro del proceso.
- Otros objetos, como descriptores de archivo abiertos.

Resumiendo, un proceso consiste en un programa, sus datos y toda la información de estado (memoria, registros, contador de programas, archivos abiertos, etc.) asociada a él.

A veces es útil permitir que múltiples hilos de control se ejecuten simultáneamente dentro de un mismo proceso. Estos hilos individuales de control se llaman hilos. De forma predeterminada, los procesos sólo tienen asociado un único hilo de texto, aunque puede ser útil tener varios. Todos los hilos de un proceso en particular comparten el mismo espacio de direcciones. En contraste, generalmente se piensa que los procesos no comparten ninguno de sus espacios de direcciones con otros procesos. Específicamente, los subprocesos (como los procesos) tienen código, memoria y otros recursos asociados con ellos.

Una gran diferencia entre los hilos y los procesos, es que las variables globales se comparten entre todos los hilos. Debido a que los subprocesos se ejecutan simultáneamente con otros subprocesos, deben preocuparse por la sincronización y la exclusión mutua al acceder a la memoria compartida.

Los hilos de Nachos, ejecutan y comparten el mismo código (el código fuente de Nachos) y comparten las mismas variables globales. El programador de Nachos, mantiene una estructura de datos llamada ready list, que mantiene un registro de los hilos listos para ejecutar. Los hilos de la de espera de ready list, están listos para ser ejecutados y pueden ser seleccionados por el programador en cualquier momento. Cada hilo tiene un estado asociado que describe lo que el hilo está haciendo actualmente.

Nachos proporciona servicios elementales para ejecutar hilos concurrentes. La clase **Thread** es la responsable de gestionar un hilo.

Cada hilo contiene información de *estado* (valores de los registros, estado de ejecución, etc.), así como un espacio de direcciones local, llamado *pila*, con las variables locales y las direcciones de retorno de subrutinas. Todos los hilos comparten las variables globales del programa.

Un hilo puede encontrarse en uno de estos cuatro estados:

READY	Lista para ejecutarse, pero encolada
RUNNING	En ejecución. Sólo puede haber un hilo en este estado.
BLOCKED	Bloqueada por haber ejecutado Thread::Sleep()
JUST_CREATED	Creada, pero todavía no está activa (no sabe qué código tiene que ejecutar)

Las funciones miembros más relevantes de la clase **Thread** son:

Thread(*"nombre"*)	constructor de la clase
Fork(*función, entero*)	lanza a ejecución el hilo (le crea una pila y la pasa a **READY**)
Yield()	cede la CPU a otro hilo

Sleep()	duerme el hilo (abandona la CPU hasta que otro hilo lo despierte)
Finish()	Finaliza el hilo y destruye su estructura interna
setStatus(*estado*)	Cambia el estado del hilo
***cadena* = getName()**	Devuelve el nombre del hilo

Cuando creamos un hilo, éste se queda en estado JUST_CREATED. El hilo no se puede ejecutar todavía, porque no tiene una pila ni sabe qué código tiene que ejecutar. Para lanzar el hilo, hay que invocar a la función Fork. Ésta recibe como primer parámetro un puntero a la subrutina donde el hilo comenzará a caminar. Cuando dicha subrutina retorne, el hilo finaliza automáticamente. La función que ejecuta el hilo debe admitir un parámetro entero, que es el segundo argumento de Fork. Este parámetro lo podemos usar para variar el comportamiento de la subrutina.

3.1.2.3. CAMBIO ENTRE HILOS

Cambiar la CPU de un hilo a otro implica suspender el hilo actual, guardar su estado, luego restaurar el estado del hilo al que se está cambiando. El interruptor de hilo finaliza en el momento en que un nuevo counter es ejecutado; en ese momento, la CPU ya no está ejecutando el código de cambio de hilo, sino que está ejecutando código asociado al nuevo hilo.

La rutina Switch (oldThread, nextThread) realiza realmente un interruptor de hilo. Switch guarda todo el estado de oldThread (oldThread es el hilo que se ejecuta cuando se llama a Switch), de forma que puede reanudar la ejecución del hilo más tarde, sin que el hilo sepa que se ha suspendido. [6]

3.1.2.4. SINCRONIZACION Y EXCLUSION MUTUA

Las rutinas de NachOS frecuentemente desactivan y activan las interrupciones para lograr la exclusión mutua. Las facilidades de sincronización se proporcionan a través de los semáforos. El objeto *Semaphore* proporciona las funciones que mencionaremos a continuación. [9]

Semaphore(char debugName, int initialValue)*

El constructor crea un nuevo semáforo de conteo que tiene un valor inicial de *initialValue*. La cadena *debugName* también está asociada con el semáforo para simplificar la depuración.

void P()

Disminuye el conteo del semáforo, cuando el conteo es cero, bloquea a la persona que realizó la llamada.

void V()

Incrementa el conteo, si alguno se encuentra bloqueado esperando el conteo, libera un hilo.

3.1.2.5. PROCESOS DE NIVEL DE USUARIO

El sistema operativo que estamos analizando, ejecuta programas de usuario en su propio espacio de direcciones privadas. Puede ejecutar cualquier binario MIPS, suponiendo que se limite solo a hacer llamadas al sistema que el sistema operativo entienda.

En Unix, los archivos *"a.out"* se almacenan en formato *"coff"*. NachOS requiere que los ejecutables estén en el formato *"Noff"* más simple. Para convertir binarios de un formato a otro, se debe usar el programa *coff2noff* .

Los archivos de formato *Noff* constan de cuatro partes. La primera parte, el Noff header, describe los contenidos del resto del archivo, dando información sobre las instrucciones del programa, las variables inicializadas y las variables no inicializadas.

El Noff header reside al principio del archivo y contiene punteros a las secciones restantes. Específicamente, el encabezado Noff contiene:

noffMagic

Un número "mágico" reservado que indica que el archivo está en formato Noff. El número mágico se almacena en los primeros cuatro bytes del archivo. Antes de intentar ejecutar un programa de usuario, NachOS comprueba el número mágico para asegurarse de que el archivo que se va a ejecutar es en realidad un ejecutable de este sistema operativo.

Para cada una de las secciones restantes, Nachos mantiene la siguiente información:

- virtualAddr: A qué dirección virtual comienza ese segmento (normalmente cero).
- inFileAddr: Puntero dentro del archivo Noff donde esa sección realmente comienza (para que Nachos pueda leerla en la memoria antes de que comience la ejecución).
- Size: El tamaño (en bytes) de dicho segmento.

Al ejecutar un programa, NachOS crea un espacio de direcciones y copia el contenido de la instrucción y los segmentos variables inicializados en el espacio de direcciones. Se debe tener en cuenta que la sección de variable no inicializada no necesita ser leída desde el archivo. Dado que se define que contiene todos los ceros, NachOS simplemente le asigna memoria dentro del espacio de direcciones del proceso del sistema operativo y lo borra. [10]

3.1.2.6. CREACION DE PROCESOS

Los procesos de NachOS se forman creando un espacio de direcciones, asignando memoria física para el espacio de direcciones, cargando los contenidos del ejecutable en la memoria física, inicializando registros y tablas de traducción de direcciones, y luego invocando *machine::Run()* para iniciar la ejecución. *Run()* simplemente pone en funcionamiento el MIPS simulado, haciendo que ingrese a un bucle infinito que ejecuta instrucciones una a una.

Stock NachOS supone que solo existe un único programa de usuario en un momento dado. Por lo tanto, cuando se crea un espacio de direcciones, NachOS supone que nadie más está utilizando la

memoria física y simplemente borra toda la memoria física. Luego, el sistema operativo lee el binario en la memoria física comenzando en la ubicación mainMemory e inicializa las tablas de traducción para hacer un mapeo uno-a-uno entre las direcciones virtuales y físicas (por ejemplo, para que cualquier dirección virtual N se corresponda directamente con la dirección física N). La inicialización de los registros consiste en ponerlos a cero por completo, configurando *PCReg* y *NextPCReg* a 0 y 4 respectivamente, y estableciendo el stackpointer a la dirección virtual más grande del proceso (la pila crece hacia abajo hacia el montón y el texto). NachOS asume que la ejecución de los programas de usuario comienza en la primera instrucción en el segmento de texto.

Cuando se ha agregado soporte para múltiples procesos de usuario, se necesitan otras dos rutinas de NachOS para el cambio de proceso. Cada vez que se suspenden los procesos actuales (p. Ej., Apropiación o reposo), el planificador invoca la rutina *AddrSpace::SaveUserState()*, para guardar adecuadamente el estado relacionado con el espacio de direcciones que las rutinas de conmutación de subprocesos de bajo nivel no conocen . Esto se vuelve necesario cuando se usa memoria virtual; cuando se cambia de un proceso a otro, se debe cargar un nuevo conjunto de tablas de traducción de direcciones. El programador de Nachos llama a *SaveUserState()* cada vez que está a punto de adelantarse a un hilo y cambiar a otro. Del mismo modo, antes de cambiar a un nuevo hilo, el programador Nachos invoca *AddrSpace::RestoreUserState.RestoreUserState()* asegura que se carguen las tablas de traducción de direcciones correctas antes de que se reanude la ejecución. [11]

3.1.2.7. LLAMADAS AL SISTEMA Y MANEJO DE EXCEPCIONES

Los programas de usuario invocan llamadas al sistema ejecutando la instrucción MIPS *"syscall"*, que genera una excepción de hardware en el núcleo de NachOS. El simulador Nachos / MIPS implementa excepciones invocando la *RoutineRaiseException()*, pasando un argumento que indica la causa exacta de la excepción. *RaiseException* , a su vez, llama a *ExceptionHandler* para encargarse del problema específico. *ExceptionHandler* recibe un solo argumento que indica la causa exacta de la trampa.

La instrucción *"syscall"* indica que se solicita una llamada al sistema, pero no indica qué llamada del sistema realizar. Por convención, los programas de usuario colocan el código que indica la llamada particular del sistema deseada en el registro r2 antes de ejecutar la instrucción *"syscall"*. Se pueden encontrar argumentos adicionales para la llamada al sistema (cuando corresponda) en los registros r4-r7, siguiendo las convenciones estándar de vinculación de llamadas al procedimiento C. Se espera que los valores de retorno de función (y llamada al sistema) estén en el registro r2 en el retorno.

Se debe tener en cuenta que, cuando se accede a la memoria del usuario desde el manejador de excepciones (o dentro de NachOS en general), las direcciones de nivel de usuario no se pueden referenciar directamente. Recuerde que los procesos a nivel de usuario se ejecutan en sus propios espacios de direcciones privados, que el kernel no puede hacer referencia directamente. Los intentos de eliminar referencias de punteros pasados como argumentos a las llamadas al sistema probablemente darán lugar a problemas (por ejemplo, fallas de segmentación) si se hace referencia directamente.[12]

3.1.2.8. INTERRUPCIONES

Las excepciones se manejan de la misma manera que las llamadas al sistema. Si un usuario la realiza la instrucción del programa causa una excepción, el simulador (Machine :: Run) llama a ExceptionHandler para que pueda ser manejado por el kernel.

En las interrupciones el simulador realiza un seguimiento del tiempo de simulación en el que el dispositivo interrumpe y se supone que ocurra.

Después de simular cada instrucción del usuario, el simulador avanza la simulación y determina si las interrupciones están pendientes desde cualquier dispositivo. Si es así, el simulador (Machine :: Run) llama al controlador del kernel para eso lo interrumpe antes de ejecutar la siguiente instrucción. Cuando el controlador del kernel retorna, la simulación continúa ejecutando instrucciones.

Como muchos dispositivos reales, los dispositivos simulados de la estación de trabajo NachOS son asíncronos, lo que significa que usan interrupciones para notificar al kernel que la operación solicitada ha sido completada, o que una nueva operación es posible.

Por ejemplo:

- La consola de entrada (teclado) genera una interrupción, cada vez que una nueva entrada está disponible
- La consola de salida (pantalla) solo puede mostrar un carácter a la vez. Eso genera una interrupción cuando está listo para aceptar otro para salida.
- El disco acepta una solicitud de lectura / escritura a la vez. Genera una interrupción cuando la solicitud ha sido completada.
- El kernel implementa interfaces síncronas para cada uno de estos dispositivos
    - implementado utilizando las primitivas de sincronización
    - las interfaces síncronas son mucho más fáciles de usar para el resto del kernel, que las interfaces asincrónicas.

3.1.2.9. SISTEMA DE ARCHIVOS

NachOS tiene dos implementaciones de sistema de archivos. El sistema de archivos real tiene una funcionalidad muy limitada. Los archivos se almacenan en disco simulado de la estación de trabajo. El sistema de archivos "stub" almacena archivos fuera de la máquina simulada, en sistema de archivos de la máquina en la que se ejecuta NachOS. En este se utiliza el sistema de archivos "stub". Es por eso que un archivo creado por este, aparece un programa de usuario NachOS en la máquina en la que se ejecuta el mismo. Esta es también la razón por la cual los programas de usuario de NachOS pueden almacenarse en archivos en el host de la máquina y no en la estación de trabajo simulada. El sistema de archivos "stub" puede parecer poco realista, sin embargo, una estación de trabajo sin disco con inicio de red utiliza un mecanismo similar.[15]

4. LABORATORIO DE SISTEMAS OPERATIVOS COMO ASIGNATURA

4.1. INTRODUCCION

En esta sección haremos la introducción del laboratorio como asignatura, mencionaremos los temas principales que brindamos como propuesta para trabajar en el desarrollo de los laboratorios durante el semestre, la metodología de trabajo dentro y fuera del aula, interacción entre el alumno con el profesor del laboratorio, nombre de la asignatura, objetivos, los prerrequisitos dentro de la malla curricular del programa, el conjunto de prácticas con su respectiva descripción, código de la asignatura, métodos de evaluación, y las fuentes bibliográficas.

4.2. NOMBRE DE LA ASIGNATURA, CODIGO Y NUMERO DE CREDITOS

- Nombre: Laboratorio de Sistemas Operativos
- Código: IS743
- Número de créditos: tres (3)
- Intensidad horaria: 4 horas semanales, además de 5 horas de trabajo fuera de clase, para un total de 144 horas en el semestre.

4.3. OBJETIVOS GENERALES

Que los estudiantes lleven a la práctica los conceptos básicos de diseño e implementación de sistemas operativos en el modelo centralizado, que conozcan las ventajas y desventajas que nos brinda un sistema operativo, como administrador de recursos, y poder abordar y conocer diversos mecanismos para brindar nuevas soluciones.

4.4. OBJETIVOS ESPECIFICOS

Que el estudiante lleve a la práctica los conceptos fundamentales en diseño y construcción de sistemas operativos tradicionales para el modelo centralizado, bajo el uso de herramientas libres, y la programación por hilos.

4.5. COMPETENCIAS GENERICAS

- Trabajo en equipo.
- Responsabilidad
- Capacidad de investigación.
- Capacidad de llevar conocimientos teóricos a la práctica
- Resolución de problemas
- Comunicación oral y escrita

4.6. COMPETENCIAS ESPECIFICAS

- Cognitivas (Saber):
    - Idioma
    - Nuevas tecnologías TIC
    - Conocimientos de arquitectura
    - Conocimiento de informática

- Procedimentales / Instrumentales (Saber hacer):
    - Redacción en interpretación de documentación técnica
    - Estimación y programación del trabajo
    - Planificación, organización y estrategia.
    - Distintos procesos de administración en recursos de máquinas

4.7. METODOLOGÍA

Proponemos como metodología de trabajo para el laboratorio, su división de trabajo en 8 prácticas, cada práctica de dos semanas de duración cada una, cumpliendo así con las 16 semanas del semestre académico, durante este tiempo, el estudiante deberá realizar las investigaciones necesarias, así como el desarrollo de la respectiva práctica, contando con el apoyo del profesor como guía, así como de las referencias bibliográficas que mencionaremos más adelante, disponibles en la biblioteca Jorge Roa Martínez de nuestra Universidad, se recomienda que los estudiantes conformen grupos de trabajo de máximo tres(3) personas, y mínimo dos(2), se recomienda no permitir que estudiantes trabajen individualmente, como fomento al trabajo en equipo, la mejora de la comunicación interpersonal, necesaria también en nuestra formación como profesionales. En cada semana de inicio de práctica, el profesor realizará la clase dando una breve introducción sobre los temas principales que se trabajarán en la práctica vigente, y las posible fuentes de consulta que propone la guía, así como fuentes de información que el profesor considere favorables para los estudiantes, en las siguientes clases de las semanas correspondientes a dicha práctica, el profesor estará disponible para las posibles consultas que el estudiante pueda tener sobre la práctica que se encuentra desarrollando.

4.8. PRERREQUISITOS

Se recomienda que el estudiante que matricule esta materia, tenga matriculada la materia IS734-Sistemas Operativos I, para su desarrollo simultáneo.

4.9. PRACTICAS

- PRACTICA 1 – Introducción a Minix 3 y NachOS
- PRACTICA 2 – Programación en SHELL
- PRACTICA 3 – Llamadas al Sistema
- PRACTICA 4 – Manejo de Procesos
- PRACTICA 5 – Manejo de Hilos

- PRACTICA 6 – Administración de Memoria
- PRACTICA 7 – Entrada y Salida
- PRACTICA 8 – Sistema de Archivos

4.10. RECURSOS

✓ Equipos de cómputo para cada estudiante.
✓ Los sistemas operativos Minix 3, y NachOS previamente instalados en las máquinas.

4.11. EVALUACIÓN

Proponemos como metodología de evaluación para las prácticas, el uso de informes escritos por cada laboratorio, donde se establezcan las técnicas usadas para el desarrollo del laboratorio, problemas presentados, metodología usada, fuentes bibliográficas consultadas, análisis y conclusiones de la práctica, por este medio, el profesor revisará el cumplimiento o no de los objetivos propuestos para la práctica.

A continuación, se muestran los porcentajes para la evaluación de la asignatura.

- Practica 1 - 12.5%
- Practica 2 - 12.5%
- Practica 3 - 12.5%
- Practica 4 - 12.5%
- Practica 5 - 12.5%
- Practica 6 - 12.5%
- Practica 7 - 12.5%
- Practica 8 - 12.5%

4.12. BIBLIOGRAFÍA

- Tanenbaum Andrew - Sistemas Operativos Análisis y Diseño
- William Stallings - Sistemas Operativos

- Carretero, Jesús - Sistemas Operativos

4.13. LABORATORIOS

4.13.1. LABORATORIO 1 - INTRODUCCIÓN A MINIX 3 Y NACHOS

4.13.1.1. INTRODUCCION

Para la interacción y práctica de los estudiantes, con las temáticas que se trabajarán durante el desarrollo del laboratorio, se buscó sistemas operativos libres, relativamente pequeños, y orientados a fines educativos, donde los estudiantes pudieran interactuar e implementar los temas que se proponen en el transcurso del laboratorio. Con esta práctica se busca que el estudiante interactúe y conozca los sistemas operativos en los que se desarrollaran los laboratorios durante el transcurso del semestre.

4.13.1.2. MARCO TEORICO

El sistema operativo Minix 3 es un sistema operativo pequeño, monolítico, de código abierto, basado en Unix, desarrollado por Andrew Stuart Tanenbaum, el autor del libro “Sistemas Operativos Análisis y Diseño”, propuesto como referencia bibliográfica para la materia Sistemas Operativos I; Es un sistema operativo que podemos considerar liviano, ya que maneja una cantidad mínima de código de ejecución en el kernel, además de manejar como procesos independientes el servidor de archivos, procesos y los controladores de hardware. Los errores en este sistema operativo pueden ser más fáciles de encontrar comparado con otros sistemas con cantidades de código exuberantes en el kernel, incluso a veces innecesarios. El hecho de que maneje los procesos de controladores de hardware como procesos independientes en el kernel, garantiza un plus de seguridad en la continuidad de ejecución, ya que los códigos externos necesarios para que el hardware funcione son desconocidos.

El sistema operativo NachOS, fue diseñado con fines educativos, lo que lo convierte en una herramienta atractiva para el laboratorio Este sistema operativo es ejecutado como un proceso de usuario en el sistema operativo anfitrión, por ende, no se puede afirmar a ciencia cierta, que sea un sistema operativo como tal, fue escrito en C++, de código abierto. Un simulador de MIPS ejecuta el código para cualquier programa de usuario que se ejecute sobre el sistema operativo NachOS [5]

4.13.1.3. OBJETIVOS

Con esta práctica se busca que el estudiante conozca los manejos básicos de los sistemas operativos Minix 3, y NachOS, su manejo de usuarios, los privilegios para usuarios, el manejo de la consola, y algunos de sus comandos de administración más básicos.

4.13.1.4. OBJETIVOS ESPECIFICOS

Se busca que el estudiante tenga interacción en los sistemas operativos Minix 3 y NachOS, en los siguientes ámbitos:

- Manejo de usuarios
- Privilegios para usuarios
- Uso de la consola (Shell)
- Comandos básicos de administración

4.13.2. LABORATORIO 2 - PROGRAMACIÓN EN SHELL

4.13.2.1. INTRODUCCION

La programación en el entorno de trabajo Shell, es una de las herramientas más poderosas para cualquier informático, aún más si hablamos de sistemas operativos libres, teniendo los permisos

de súper usuario, se podría afirmar coloquialmente que “podemos hacer cualquier cosa”. Esta práctica está enfocada a que el estudiante conozca e interactúe con esta herramienta, muy utilizada por los administradores de sistemas de información, en el sistema operativo Minix 3 implementando y ejecutando scripts, variables y archivos.

4.13.2.2. MARCO TEORICO

Aunque no forma parte del sistema operativo, utiliza con frecuencia muchas características del mismo, es la interfaz principal entre un usuario sentado en su terminal y el sistema operativo, a menos que el usuario esté usando una interfaz gráfica de usuario. Existen muchos shells, incluyendo sh, csh, ksh y bash. Todos ellos soportan la funcionalidad antes descrita, que se deriva del Shell original (sh), Minix 3 utiliza un Shell muy similar a bash. Cuando cualquier usuario inicia sesión, se inicia un Shell. El Shell tiene la terminal como entrada estándar y salida estándar. Empieza por escribir el indicador de comandos (prompt), un carácter tal como un signo de dólar, que indica al usuario que el Shell está esperando aceptar un comando. [8]

4.13.2.3. OBJETIVOS

Con esta práctica se busca que el estudiante se familiarice con el Shell del sistema operativo, conozca sus funcionalidades, y una variedad de comandos que le permitan sacar el mayor provecho al sistema operativo, al mismo tiempo que le permite interactuar con el mismo de una manera más amena.

4.13.2.4. OBJETIVOS ESPECIFICOS

Se busca que el estudiante en esta práctica adquiera conocimientos en la programación en Shell en los siguientes ámbitos:

Manejo de directorios

- Redirección de Entradas / Salidas
- Línea de Órdenes
- Variables

4.13.3. LABORATORIO 3 - LLAMADAS AL SISTEMA

4.13.3.1. INTRODUCCION

Como usuario, en su mayoría programadores, es casi imposible no necesitar en algún momento de recursos y funciones del sistema operativo, en el proceso de desarrollo de software por ejemplo, se necesitará conocer los tipos de llamadas e interacción que tendrá el software que se implementará, con el sistema operativo donde se implantará, que tipo de recursos necesita, y cómo puede acceder a ellos, por ende esta práctica está desarrollada para que el estudiante conozca cómo puede acceder a recursos y funcionalidades del sistema operativo, y en qué caso podría necesitarlos.

4.13.3.2. MARCO TEORICO

Las llamadas al sistema, son servicios o recursos que necesita el usuario del sistema operativo, y realiza esta solicitud al kernel, por medio de llamadas como open(), read(), fork(), socket, entre otros. Dado que el acceso a ciertos recursos del sistema requiere la ejecución de código en modo privilegiado, el sistema operativo ofrece un conjunto de métodos o funciones que el programa puede emplear para acceder a dichos recursos. En otras palabras, el sistema operativo actúa como intermediario, ofreciendo una interfaz de programación (API) que el programa puede usar en cualquier momento para solicitar recursos gestionados por el sistema operativo. [14]

Cuando se ejecuta un programa se copia el ejecutable en una imagen de programa (imagen de carga) en la memoria principal y se le asigna un espacio de direcciones. Cada programa en ejecución es un proceso que es manejado por el sistema mediante una estructura de datos a la que se accede a través de un entero denominado identificador de proceso (pid). La familia de llamadas exec ( ), reemplaza la imagen del proceso actual con una nueva imagen de carga

referente al archivo nombrado en el primer parámetro. En el caso más general, exec tiene tres parámetros: el nombre del archivo por ejecutar, un apuntador al arreglo de argumentos y un apuntador al arreglo del entorno. Típicamente, después de una bifurcación (fork), uno de los dos procesos emplea la llamada exec, para reemplazar su espacio de memoria virtual con un nuevo programa. Esta llamada carga en memoria un archivo binario (destruyendo el contenido en memoria del programa que contiene la llamada, y comienza su ejecución. Se dispone de varios procedimientos de biblioteca, execl, execlp, execle, exect, execv, execve, y execp, para que los parámetros sean omitidos o especificados de varias formas.

4.13.3.3. OBJETIVOS

Con esta práctica se busca que el estudiante conozca y aplique las llamadas más importantes al sistema operativo, que conozca e interactúe con esta herramienta como parte esencial de su formación en esta área de conocimiento.

4.13.3.4. OBJETIVOS ESPECIFICOS

Con esta práctica se busca que el estudiante adquiera y lleve a la práctica conocimientos en los siguientes temas:

- Llamadas de acceso a ficheros
- Llamadas de control de procesos
- Llamadas de comunicación entre procesos.

4.13.4. LABORATORIO 4 - MANEJO DE PROCESOS

4.13.4.1. INTRODUCCION

El concepto más importante en cualquier sistema operativo es el de proceso, una abstracción de un programa en ejecución; todos los demás conceptos que veremos en el laboratorio, así como todos los demás que abarcan la temática teórico-práctica de los sistemas operativos, dependen de este concepto, como usuarios hemos interactuado desde hace años con procesos en los distintos sistemas operativos que hemos tenido la oportunidad de operar, solo que no lo sabíamos, cada programa que hemos ejecutado, son conocidos como procesos, por lo cual es importante que el estudiante tenga una comprensión profunda y clara acerca de la información teórica de lo que es un proceso, así como poder interactuar de manera práctica con estos en los sistemas operativos utilizados dentro del laboratorio.

4.13.4.2. MARCO TEORICO

Los procesos son una de las abstracciones más antiguas e importantes que proporcionan los sistemas operativos, proporcionan la capacidad de operar concurrentemente, incluso cuando hay sólo una CPU disponible. Convierten una CPU en varias CPU virtuales. Sin la abstracción de los procesos, la computación moderna no podría existir. [8]

Un proceso puede pasar por varios estados durante su ejecución. En la Figura 2, se muestra un diagrama de los posibles estados que puede tener un proceso.

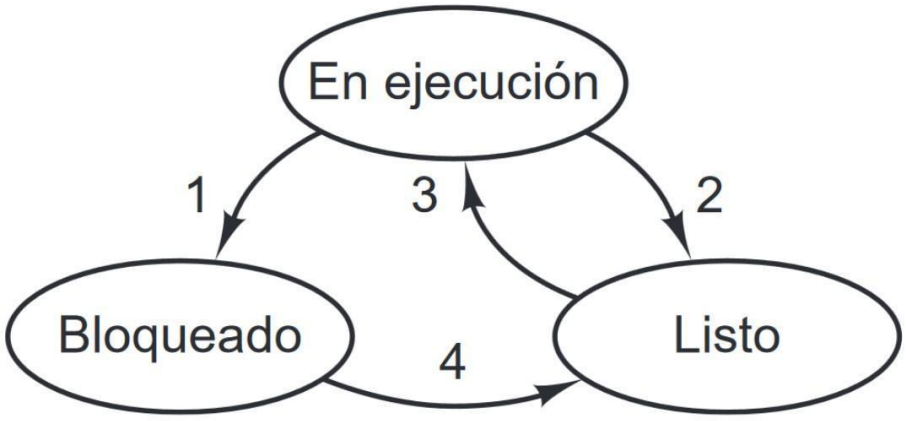

Figura 2: Estados de un proceso [8]

El orden de los estados según los números de la figura serían los siguientes:

1. El proceso se bloquea para recibir entrada

2. El planificador selecciona otro proceso
3. El planificador selecciona este proceso
4. La entrada ya está disponible

4.13.4.3. OBJETIVOS

Con esta práctica se busca que el estudiante lleve a la práctica los conceptos teóricos que obtiene en la materia teórica de sistemas operativos referente al manejo de procesos, conozca sus funciones, su implementación y el manejo de sus estados.

4.13.4.4. OBJETIVOS ESPECIFICOS

Se busca que el estudiante adquiera conocimientos llevados a la práctica en los siguientes ámbitos:

- El modelo del proceso
- Creación de Procesos
- Terminación de Procesos
- Jerarquías de Procesos
- Estados de un proceso
- Implementación de los procesos
- Comunicación entre procesos

4.13.5. LABORATORIO 5 - MANEJO DE HILOS

4.13.5.1. INTRODUCCION

Para los estudiantes de la facultad, como parte de su formación en el campo de desarrollo de software, es fundamental e indispensable en su formación como profesionales competitivos en el mundo laboral, que conozcan las ventajas y desventajas del uso de hilos de ejecución, y es

responsabilidad del programa, brindarle al estudiante conocimiento teórico, ligado del conocimiento práctico de esta temática, por ende, decidimos incluir esta temática en nuestra propuesta curricular, además de fomentarla como una de las prácticas principales del laboratorio que traemos como propuesta.

4.13.5.2. MARCO TEORICO

En los sistemas operativos tradicionales, cada proceso tiene un espacio de direcciones y un solo hilo de control. De hecho, ésa es casi la definición de un proceso. Sin embargo, con frecuencia hay situaciones en las que es conveniente tener varios hilos de control en el mismo espacio de direcciones que se ejecuta en cuasi-paralelo, como si fueran procesos (casi) separados (excepto por el espacio de direcciones compartido). La principal razón de tener hilos es que en muchas aplicaciones se desarrollan varias actividades a la vez. Algunas de ésas se pueden bloquear de vez en cuando. Al descomponer una aplicación en varios hilos secuenciales que se ejecutan en cuasi-paralelo, el modelo de programación se simplifica. Los hilos son útiles en los sistemas con varias CPUs, en donde es posible el verdadero paralelismo [8]

4.13.5.3. OBJETIVOS

Con esta práctica se busca que el estudiante tenga una interacción directa con el manejo de hilos, como herramienta fundamental en el uso de sistemas operativos, y como herramienta de apoyo a los diseños de soluciones de software, además como una herramienta de apoyo como antelación para la asignatura sistemas distribuidos

4.13.5.4. OBJETIVOS ESPECIFICOS

- Implementación de hilos
- Hilos en el kernel
- Hilos en POSIX

4.13.6. LABORATORIO 6 - ADMINISTRACIÓN DE MEMORIA

4.13.6.1. INTRODUCCION

La administración de memoria es una tarea realizada por el sistema operativo que consiste en gestionar la jerarquía de memoria, en cargar y descargar procesos en memoria principal para que sean ejecutados. Para ello el sistema operativo gestiona lo que se conoce como MMU o Unidad de Administración de Memoria, el cual es un dispositivo hardware que transforma las direcciones lógicas en físicas.

Su trabajo es seguir la pista de qué partes de la memoria están en uso y cuáles no lo están, con el fin de poder asignar memoria a los procesos cuando la necesiten, y recuperar esa memoria cuando dejen de necesitarla, así como gestionar el intercambio entre memoria principal y el disco cuando la memoria principal resulte demasiado pequeña para contener a todos los procesos

4.13.6.2. MARCO TEORICO

Minix 3 es basado en Unix, y todo proceso en Unix tiene un espacio de direcciones que consiste lógicamente en tres segmentos: texto, datos y pila. En la figura 3, se muestra un espacio de direcciones de un proceso de ejemplo, como el proceso A. El segmento de texto contiene las instrucciones de máquina que forman el código ejecutable del programa. Es producido por el compilador y el ensamblador al traducir el programa de C, C++ u otro lenguaje en código máquina. Por lo general, el segmento de texto es de sólo lectura. Los programas automodificables quedaron obsoletos debido a que era muy difícil comprenderlos y depurarlos. Por lo tanto, el segmento de texto no aumenta ni disminuye, ni cambia de cualquier otra forma. [8].

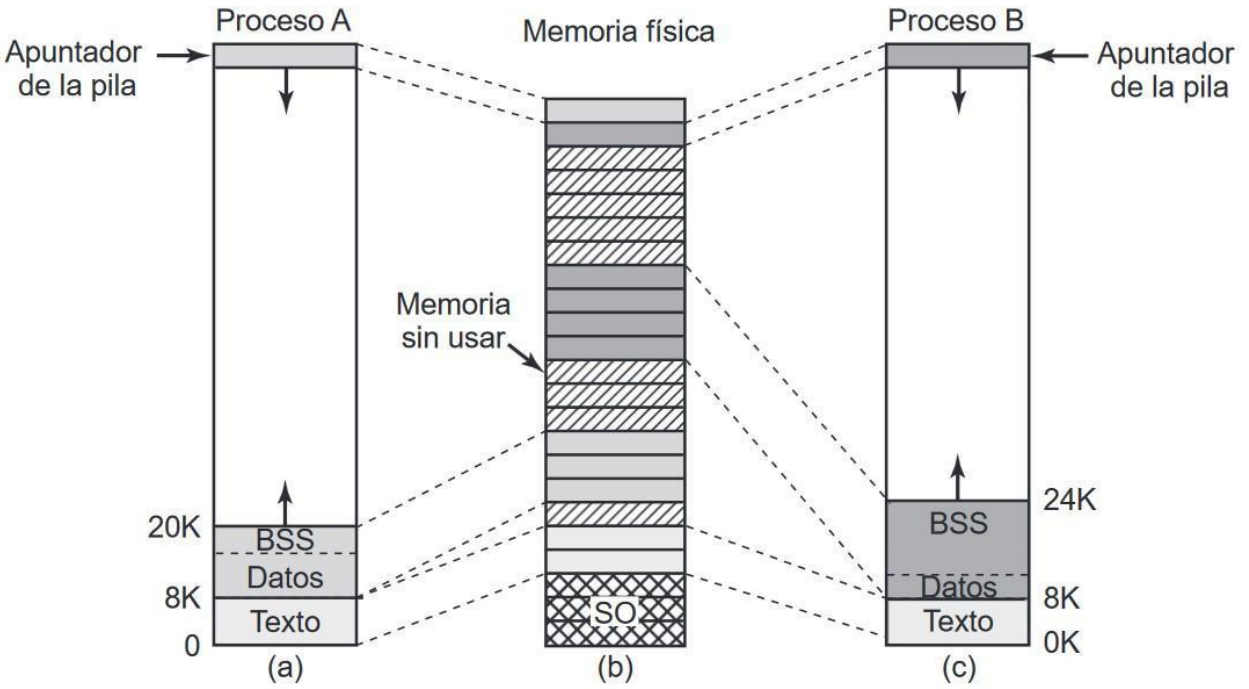

Figura 3 Ejemplo espacios de direcciones

En la figura anterior, (a), corresponde al espacio de direcciones virtuales del proceso A, (b), a la memoria física, y (c), al espacio de direcciones virtuales del proceso B.

El segmento de datos contiene almacenamiento para todas las variables del programa, cadenas, arreglos y demás datos. Tiene dos partes, los datos inicializados y los datos sin inicializar. Por cuestiones históricas, estos últimos tipos de datos se conocen como el BSS (conocido históricamente como Bloque iniciado mediante un símbolo). La parte inicializada del segmento de datos contiene variables y constantes del compilador que necesitan un valor inicial cuando empieza el programa. Todas las variables en la parte del BSS se inicializan con cero después de cargarlas. Por ejemplo, en C es posible declarar una cadena de caracteres e inicializarla al mismo tiempo. Cuando inicia el programa, éste espera que la cadena tenga su valor inicial. Para implementar esta construcción, el compilador asigna a la cadena una ubicación en el espacio de direcciones, y asegura que cuando se inicie el programa esta ubicación contenga la cadena apropiada. Desde el punto de vista del sistema operativo, los datos inicializados no son tan distintos del texto del programa; ambos contienen patrones de bits producidos por el compilador, que se deben cargar en memoria al iniciarse el programa. [8].

4.13.6.3. OBJETIVOS

Con esta práctica se busca que el estudiante tenga una interacción directa con el manejo y administración de memoria y el manejo de direcciones virtuales, como herramienta fundamental en el uso de sistemas operativos, y como herramienta de apoyo a los diseños de soluciones de software, además como una herramienta de apoyo como antelación para la asignatura sistemas distribuidos.

4.13.6.4. OBJETIVOS ESPECIFICOS

- Llamadas al sistema de administración de memoria
- Manejo de direcciones virtuales

4.13.7. LABORATORIO 7 - ENTRADA Y SALIDA

4.13.7.1. INTRODUCCION

La mayoría de computadores que conocemos, (por no decir todos), tienen algún sistema de E/S, de datos, y por ende el sistema operativo como interfaz entre el usuario y el hardware, es responsable de garantizar el correcto flujo de datos entre los programas que está ejecutado el usuario que necesitan de estos datos, y el dispositivo externo al sistema

4.13.7.2. MARCO TEORICO

En el sistema de E/S de Minix, todos los dispositivos son vistos y trabajados como archivos, y utiliza las llamadas al sistema *read* y *write*, usados para acceder a los archivos ordinarios del sistema. Estos archivos pueden ser usados como cualquier otro, utilizando las llamadas al sistema usuales como *open*, *read* y *write*.

Por ejemplo, se puede enviar un archivo directamente a imprimir, simplemente usando el comando cp, copiando el archivo a la impresora.

4.13.7.3. OBJETIVOS

Con esta práctica se busca que el estudiante lleve a la práctica los conceptos teóricos que obtiene en la materia teórica de sistemas operativos referente a llamadas al sistema y accesos y funcionamiento de archivos.

4.13.7.4. OBJETIVOS ESPECIFICOS

- Llamadas al sistema
- Acceso a archivos

4.13.8. LABORATORIO 8 – SISTEMA DE ARCHIVOS

4.13.8.1. INTRODUCCION

Todos sabemos que los dispositivos de computación actuales tienen capacidad de almacenar grandes cantidades de información, y que los antiguos, aunque en poca capacidad, también tenían la obligación de manejar y almacenar información para su correcto funcionamiento, dado que todos los procesos en ejecución en una sistema operativo tiene la necesidad de pedir y entregar información al sistema operativo, ya sea información temporal, o información que es indispensable que el sistema operativo apoyado en el hardware de la máquina, almacene indefinidamente para su posterior consulta, dicha información se le conoce como archivo, en la actualidad se cuenta con una gran variedad de dispositivos de almacenamiento para almacenar dichos archivos, como discos duros, memorias USB, cds, cintas magnéticas, discos ópticos, entre muchos otros. En palabras más formales, los archivos son unidades lógicas de información creados por los procesos, procesos que pueden acceder, modificar, e incluso crear nuevos archivos a partir de los ya existentes; Dicha información debe ser independiente a los posibles

errores que pueda tener el sistema operativo y/o el hardware. El sistema operativo es responsable de cómo se estructuran, denominan, abren, utilizan, protegen, implementan y administran los archivos, a todo este manejo se le conoce como sistema de archivos del sistema operativo, y es el tema que vamos a manejar en este laboratorio.

4.13.8.2. MARCO TEORICO

El sistema de ficheros minix fue diseñado para ser usado con Minix, el sistema operativo diseñado por Tanenbaum como apoyo a la docencia. Este sistema de ficheros copia las estructuras básicas del Unix File System, pero, debido a la naturaleza del sistema minix, elimina algunas características complejas con el fin de mantener claro y simple el código fuente.

Un sistema de archivos MINIX tiene seis componentes:

El Bloque de arranque, el cual es el que se almacena en el primer bloque de un disco duro. Contiene el cargador de Arranque, que carga y corre un sistema operativo dentro del sistema conocido como sistema de inicio.

El segundo bloque es el Superblock, el cual almacena datos relacionados al sistema de archivos, que permite al sistema operativo localizar y entender otras arquitecturas de sistemas de archivos. Por ejemplo, el número de nodos y zonas, el tamaño de los dos bitmaps y el bloque de inicio de un área de datos.

El inode bitmap, es el mapa simple de uno de los inodos que rastrea cuales están en uso y cuáles están libres, representándolos con un 1 (para los en uso) y con un 0 (para los libres).

El zone bitmap, trabaja de similar forma al inode bitmap, excepto que rastrea las zonas.

El área de inodos. Cada archivo o directorio es representado como un inodo, el cual graba metadatos incluyendo los tipos (archivo, directorio, bloque, carácter, pipe), identidad (is) para el usuario y el grupo, tres registros de fecha y el tiempo de último acceso, la última modificación y el último cambio de estado. Un inodo también contiene una lista de las

direcciones que indican las zonas en el área de datos donde el archivo o los datos de directorio están actualmente almacenados.

El data area (área de datos), es el componente más largo de un sistema de archivos, que usa la mayor parte del espacio. Es donde los archivos y directorios de datos están almacenados.

4.13.8.3. OBJETIVOS

Con esta práctica se busca que el estudiante lleve a la práctica los conceptos teóricos que obtiene en la materia teórica de sistemas operativos referente al manejo archivos, sus componentes y funcionamiento de los mismos.

4.13.8.4. OBJETIVOS ESPECIFICOS

Se busca que el estudiante en esta práctica refuerce su conocimiento teórico y adquiera conocimiento práctico en los siguientes temas:

- Permisos de archivos
- Tipos de archivos

5. CONCLUSIONES

Luego de realizar la respectiva investigación sobre las herramientas Minix3 y NachOS, en cuanto a conceptos, usos y aplicaciones en los sistemas operativos, llegamos a la conclusión que estas herramientas son de vital importancia a la hora de hablar de sistemas operativos, para fines educativos, ya que fueron realizados y diseñados para tal fin, permitiéndoles así a los estudiantes adquirir conceptos más claros y concretos sobre los sistemas operativos, por medio de las prácticas de laboratorio propuestas como proyecto de grado, las cuales les permiten afrontar con mejores bases las materias posteriores a la asignatura sistemas operativos I.

BIBLIOGRAFIA

[1] Installing MINIX 3 Fuente: http://wiki.minix3.org/doku.php?id=usersguide:doinginstallation

[2] Minix/ARM. Fuente: http://wiki.minix3.org/doku.php?id=developersguide:minixonarm

[3]Minix 3/Download Fuente: http://wiki.minix3.org/doku.php?id=www:download:start

[4] Minix 3 Sobre Arquitectura ARM, J. Adrián Bravo Navarro Héctor Cortiguera Herrera Jorge Quintás Rodríguez, Universidad Complutense de Madrid.

[5] Wikiwand NachOS Fuente: http://www.wikiwand.com/es/NachOS

[6] Mechanics of Thread Switching, Thomas Narten Fuente: https://users.cs.duke.edu/~narten/110/nachos/main/node13.html#SECTION00041000000000000000

[7] Nachos Threads, Thomas Narten, Fuente: https://users.cs.duke.edu/~narten/110/nachos/main/node12.html

[8] Sistemas Operativos Modernos 3 edicion - Andrew S Tanembaw

[9] Synchronization and Mutual Exclusion, Thomas Narten, Fuente: https://users.cs.duke.edu/~narten/110/nachos/main/node15.html#SECTION00043000000000000000

[10] User-Level Processes, Thomas Narten, Fuente: https://users.cs.duke.edu/~narten/110/nachos/main/node17.html#SECTION00050000000000000000

[11] Process Creation, Thomas Narten, Fuente: https://users.cs.duke.edu/~narten/110/nachos/main/node18.html#SECTION00051000000000000000

[12] System Calls and Exception Handling, Thomas Narten Fuente: https://users.cs.duke.edu/~narten/110/nachos/main/node20.html#SECTION00053000000000000000

[13] Sistemas operativos NachOS introducción, Jonathan Makuc Fuente: https://www.google.com.co/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&ved=0ahUKEwijpprQ577XAhWIKCYKHZ_cDvAQFggyMAI&url=http%3A%2F%2Fcmz.ublog.cl%2Farchivos%2F549%2Funab_nachos_01_introduccion.pdf&usg=AOvVaw0TnPRam1oI1aod_03J1M9c

[14]Llamadas al sistema, wiki de la asignatura de Sistemas Operativos del departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla, Fuente: http://1984.lsi.us.es/wiki-ssoo/index.php/Llamadas_al_sistema

[15] Who is NachOS?
https://www.student.cs.uwaterloo.ca/~cs350/W07/notes/nachos.pdf